SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE
J. Softw. Maint: Res. Pract. **10**, 203–224 (1998)

Practice

Framework for Tool Evaluation for a Maintenance Environment

MATIAS VIERIMAA[1], JORMA TARAMAA[1,*], HELI PUUSTINEN[1], KATI SUOMINEN[2] and TOMMI KETOLA[2]

[1]*VTT Electronics, PO Box 1100, FIN–90571 Oulu, Finland*
[2]*Space Systems Finland Ltd., PO Box 324, FIN–02151 Espoo, Finland*

SUMMARY

Tool evaluation forms an essential part of tool development. Unfortunately, there are only a few solutions for this activity. As part of the AMES project, tool evaluation was implemented as part of a more comprehensive framework, used with the development and application of several AMES tools for application maintenance. These included an application understanding tool-set, a disabbreviation tool, a reverse-engineering tool and an impact analysis tool-set. In addition, the requirement for applicability both to on-board space software and to tool software has directed our approach in setting requirements for tools evaluation.

The central element of the evaluation process was an AMES-developed framework including evaluation criteria and the use of a goal/question/metrics (GQM) based approach. The criteria have produced detailed information about each tool. The final result can be regarded as a combination of the score for each criterion and explicit metrics data. In addition, descriptive information about applications was used in tool evaluation. © **1998 John Wiley & Sons, Ltd.**

KEY WORDS: software tool evaluation; application management; software maintenance; space applications; GQM; tool evaluation criteria

1. INTRODUCTION

The value of case studies of software tool evaluation has been well established (Kitchenham, Pickard and Pfleeger, 1995). Also, evaluation methods and frameworks have been studied (Kitchenham, 1996; Boloix and Robillard, 1995), but their procedures are often difficult to use in evaluation work at the practical level.

The AMES (see Appendix A for a glossary of acronyms) project needed software tools to support applications maintenance and management. AMES was ESPRIT project number 8156. The partners were Cap Gemini Innovation from France, Cap Programator from

* Correspondence to: Jorma Taramaa, VTT Electronics, PO Box 1100, FIN–90571 Oulu, Finland. E-mail: Jorma.Taramaa@vtt.fi

Contract/grant sponsor: EU Information Technology Programme ESPRIT III; contract/grant number: EP8156
Contract/grant sponsor: TEKES

CCC 1040–550X/98/030203–22$17.50

Received 15 November 1996
Revised 20 March 1998

Sweden, University of Durham from the United Kingdom, Intecs Sistemi Spa from Italy, Matra Marconi Space France from France, OPL-TT from Spain, Space Systems Finland from Finland and VTT Electronics from Finland. The AMES project ran from 1993 to 1996, with action to develop tools to support applications management occurring in the final phases.

The tools developed and hence candidates for evaluation included the application understanding tool-set (Boldyreff *et al.*, 1995; Vierimaa, 1996, pp. 44–64), the disabbreviation tool (Laitinen, 1995, pp. 78–81), the reverse engineering tool-set (Battaglia and Savoia, 1997), and the impact analysis tool-set (Barros *et al.*, 1995). In addition, the inclusion of two on-board space systems and a software tool as applications to which the tools should be applicable, have directed our approach in setting requirements for tools evaluation. The three main factors in our tool evaluation approach were application-specific characteristics, the scores on a three-valued scale of evaluation criteria and some absolute metrics.

The evaluation criteria used included data from performance measures on acid tests. Unfortunately, a simple scoring did not provide a sufficient view of the tools' different features to be evaluated. Therefore we extended the evaluation by using metrics based on the goal/question/metrics (GQM) paradigm (Basili and Weiss, 1984; Basili, Caldiera and Rombach, 1994).

2. EVALUATION AS PART OF APPLICATION MANAGEMENT

2.1. Application management

In the AMES project, we placed developing tool evaluation in the context of a larger framework, as part of the development of application management combining maintenance strategies with maintenance processes. Application management had been introduced into the AMES project as follows (AMES, 1994, p. 6):

'Contractual responsibility for the management and execution of all activities related to the maintenance of existing applications.'

In addition, we have related application management to a comprehensive framework of configuration management practices as one step in that comprehensive framework, which ranges from version control to product level management (Taramaa, Seppanen and Makarainen, 1996).

Tool evaluation had an essential role in the final phase of the tool development work. In it, the tool developers got their first usage experience, and the AMES project tool users had their first practical views of various usage possibilities, as diagrammed in Figure 1. The specification of the tool evaluation process was made in terms of scenarios that described the experiments to be done with each tool. A prior tool specification phase had provided a basis for the tool development, and given a preliminary overview of the evaluation requirements. In the assessment phase it was possible to apply the tools to various applications with different requirements and to use the tools in different combinations.

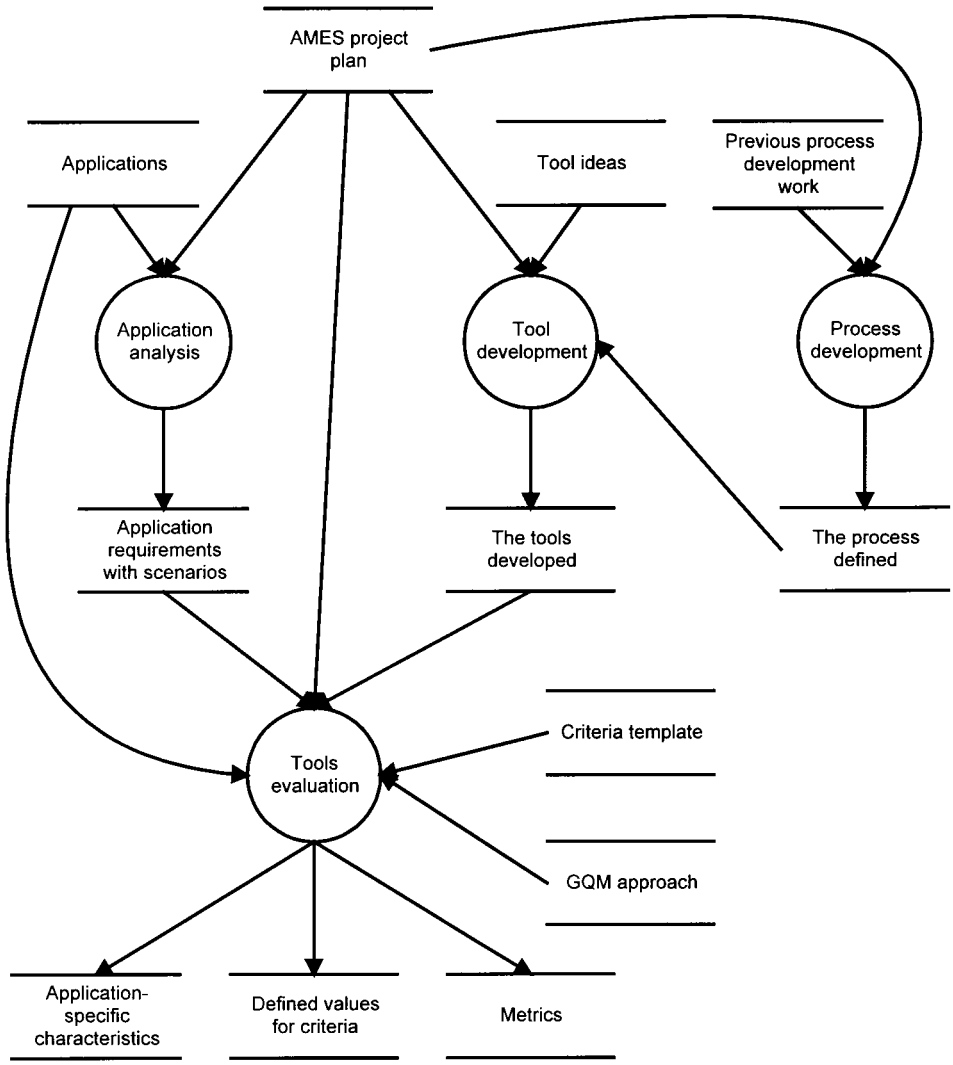

Figure 1. Tools evaluation as a part of the AMES project

A process model of application management has been developed as part of the AMES project, where it has been defined as a three-level framework (AMES, 1994, pp. 9–13) which has been related to other corresponding models, such as Boldyreff, Burd and Hather (1994). The levels are:

- strategic,
- management, and
- technical.

The strategic and management levels provide definitions for the activities that support the activities of technical levels, such as management of cost, resources, project practices and customer relations in various maintenance and evolution situations. Figure 2 summarizes the separate steps at the technical level for maintenance and evolution. The process model was implemented partly by using a process enactment tool, ProcessWeaver, with which various potential maintenance situations and tools needed for separate subphases were analysed (Taramaa, Makarainen and Ketola, 1995; Taramaa and Ketola, 1995). The definition of the technical level is based on the work done in the ESF/EPSOM project (Harjani and Queille, 1992). The ESF/EPSOM (European Platform for Software Maintenance, 1991–1994) was a subproject of the Eureka Software Factory project (ESF) of the Eureka programme. Partners were Matra Marconi Space France, Cap Gemini Innovation, and Electricite de France.

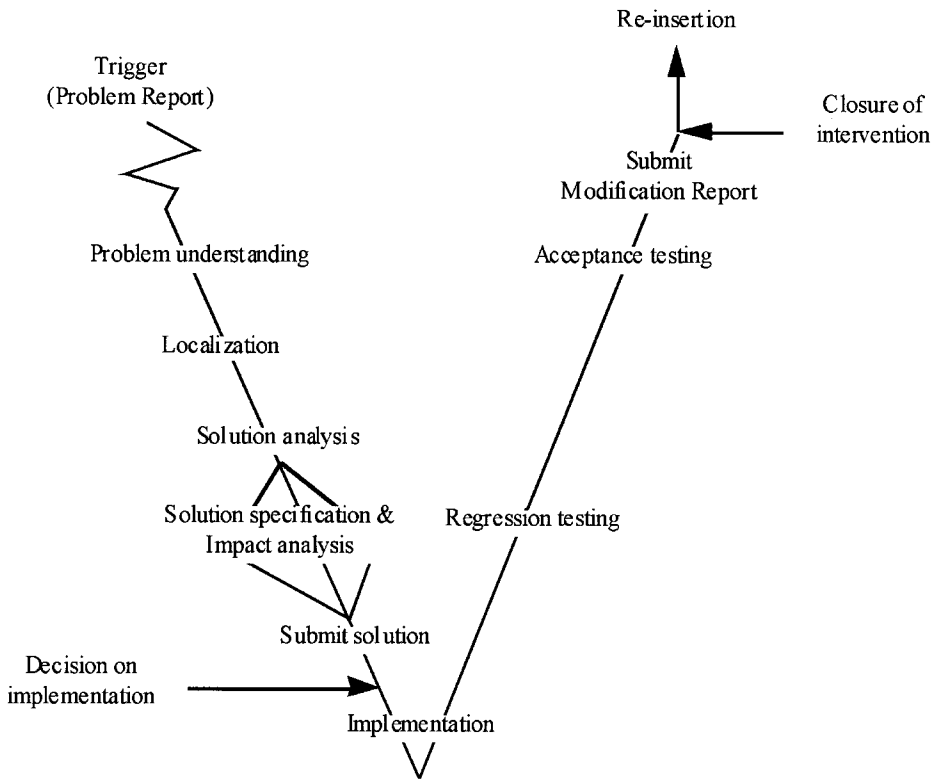

Figure 2. AMES general process for application management on the technical level (Barros et al.*, 1995, p. 44). Figure originally published in Harjani, D.-R. and Queille, J.-P. (1992) 'A process model for the maintenance of large space systems software', in* Proceedings of the Conference on Software Maintenance, *IEEE Computer Society Press, Los Algmitos CA, p. 128. Reproduced with the kind permission of the IEEE*

2.2. Application analysis and requirements produced

Application management needs and current practices were examined from two viewpoints:

- the whole organization, and
- one selected, typical software project.

Ishikawa diagrams were used as a tool to structure the work. A specific Ishikawa diagram defined in the ESF/EPSOM project (Haziza *et al.*, 1992, p. 20) was used to prioritize the factors affecting the quality of maintenance. Figure 3 is an example that diagrams that application management accounts for only part of the factors affecting software maintenance quality. In the AMES project we concentrated only on aspects that can be categorized as belonging to application management—i.e., either maintenance strategy or maintenance management.

The selected software project dealt with an on-board real-time application system and its maintenance and evolution. This application, called METEGG, was using space-specific design aids, such as the CASE tool HOOD, a semi-formal design method widely adopted in the European space industry (Robinson, 1992, pp. 1–10). In addition the software development was subject to aerospace software standards which had other effects, especially on documentation.

The analysis we conducted produced a set of specific requirements related to application management (Makarainen 1996, pp. 37–41):

- understanding, redocumenting and maintaining C code at the source-code level,
- understanding and maintaining documentation,
- understanding and maintaining a HOOD design, and
- reverse engineering C source code for further maintenance at HOOD and/or C source code levels.

3. EVALUATION FRAMEWORK

3.1. Two parts

The evaluation is based on a framework that includes:

- criteria, and
- goals with explanatory questions and specific metrics.

The criteria themselves are not sufficient, since the justifications are vague enough to require interpretation. Therefore, the goal/question/metrics (GQM) paradigm (Basili and Weiss, 1984; Basili, Caldiera and Rombach, 1994) was used to provide a more explicit justification for the tools evaluation. The results of the GQM analysis consist of various values that are related to the criteria. The relationship between the criteria and the metrics

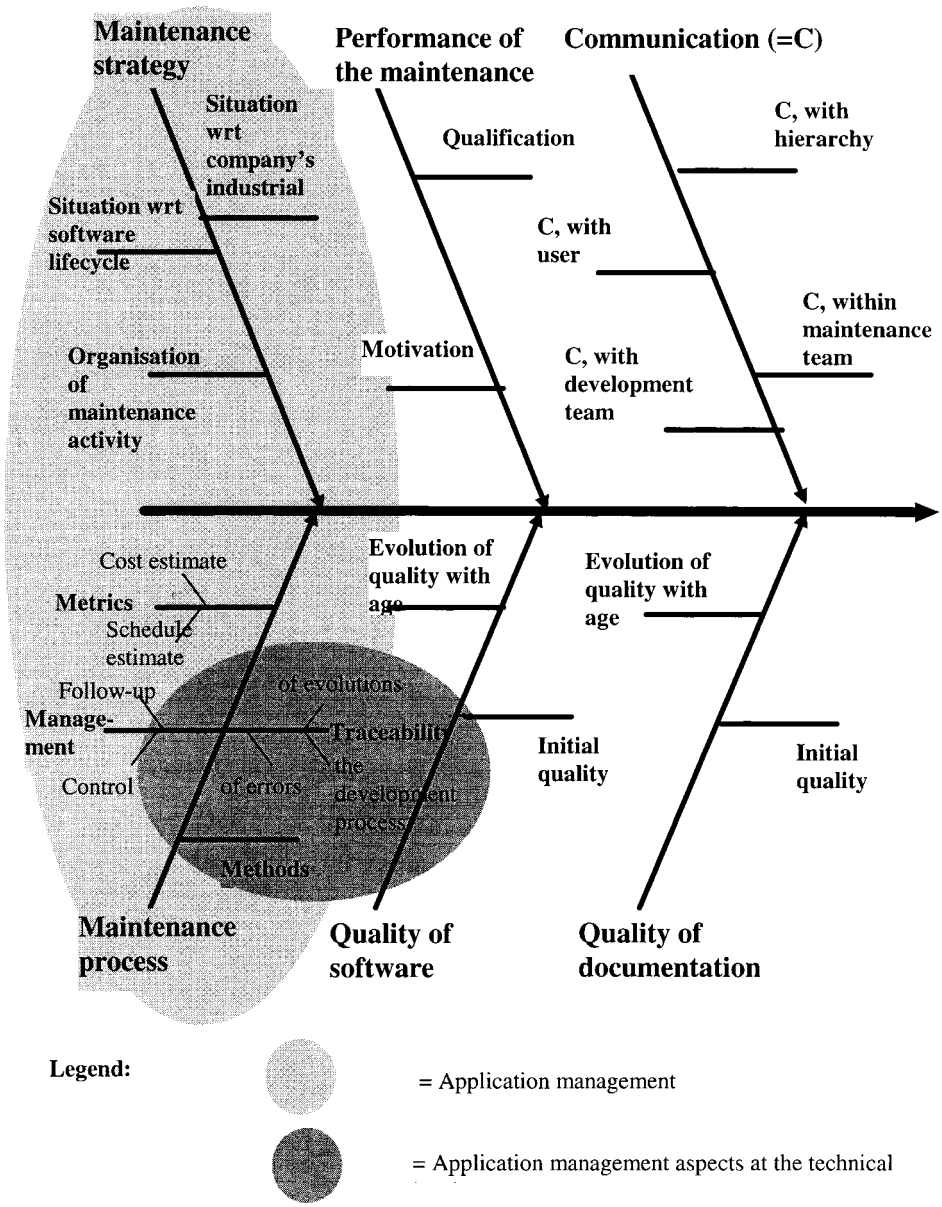

Figure 3. Main quality factors for software maintenance (Haziza et al.*, 1992, p. 20, shading added). Figure originally published in Haziza, M., Voidrot, J. F., Minor, E., Pofelski, L. and Blazy, S. (1992) 'Software maintenance: an analysis of industrial needs and constraints', in* Proceedings of the Conference on Software Maintenance, *IEEE Computer Society Press, Los Alamitos CA, pp. 20. Reproduced with the kind permission of the IEEE*

is an *n*:*m* relation—i.e., any one metric can justify several criteria, and any one criterion can use several metrics as justification. This type of relationship is diagrammed in Figure 4.

3.2. Evaluation criteria

The evaluation used seven criteria:

- performance,
- ease of use,
- tool integration/data exchange,
- robustness,
- functionality,
- training courses, and
- documentation.

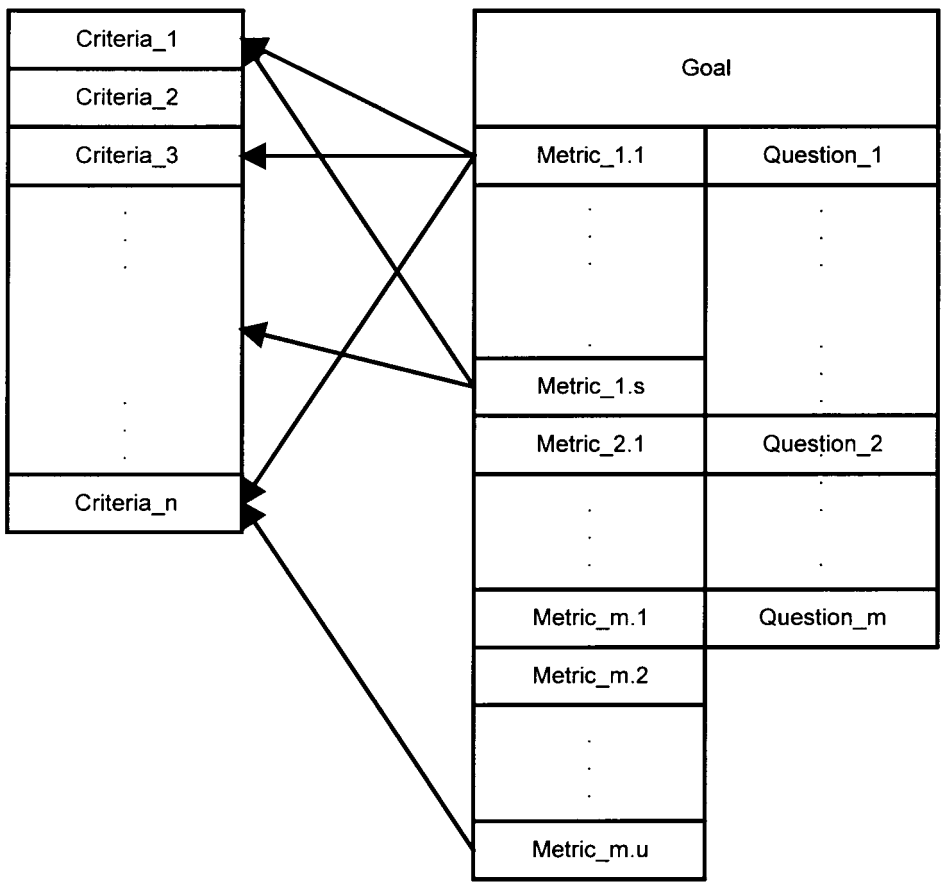

Figure 4. The tools evaluation framework of the AMES project

A more detailed description is presented in Appendix B. In scoring the criteria, we have recognized three different values or levels:

- *Low* indicates that the tool does not have this function/property, or it is poor.
- *Normal* indicates that the tool has the function/property in question, but it is hard to use and results from using it do not encourage continuous usage. Sometimes this value was influenced by the maintainers, who did not find that specific part of the tool useful, even though the function/property existed in the tool.
- *Good* indicates that the tool has this property/function and it is well supported.

3.3. Goal/question/metrics in tool evaluation

The main difficulty in applying these three levels is to find a clean way of distinguishing each item's level or value. Therefore we applied the GQM paradigm to supply a formal justification of the levels for each criterion.

The GQM approach has been successfully applied in process measurement, as reported in the literature (NASA, 1995, pp. 5–20; Oivo, 1994, pp. 62–64; van Latum *et al.*, 1998). The GQM approach supports the operational definitions of all kinds of measurement goals due to its top–down refinement into metrics via questions. The GQM plan contains:

- a goal which is defined according to a template with different dimensions, such as object, purpose, quality focus, viewpoint, and context of measurement,
- a set of questions refining the goal, and
- a set of metrics associated with every question in order to answer it quantitatively.

Table 1 shows the use of the GQM approach's basic template used for defining metrics. Within the instantiation of the basic statement 'Analyse' we sometimes particularized the instantiation to an AMES tool, such as AU, RN and IAS (see Appendix A for an acronym glossary).

The criteria were used to design questions and metrics for the tool evaluation—i.e., we created a list of questions and the metrics related to them. Table 2 describes all the questions and the metrics associated with each question. 'Qn' identifies the question number 'n', and 'Mn.k' identifies the metric number 'k' within the question 'n'.

There are application-specific and common elements in these questions and metrics. Examples of application-specific questions and metrics are shown in Table 2 for the AU,

Table 1. The basic GQM template for goal definition

Basic statement	Instantiation
analyse	the products (AU, RN, IAS)
in order to	evaluate them
with respect to	the tool value
from the viewpoint of	the maintainer

Table 2. GQM questions and metrics for tool evaluation

Question	Metric	Definition of question or metric
Q1		What do the tools require from the system?
	M1.1	Disk space required for the tool
	M1.2	Disk space required for additional tools
	M1.3	Resource usage
	M1.4	Memory usage
	M1.5	Number of available platforms
Q2		What do the tools require from the user during installation?
	M2.1	Number of parameters needed to be configured
Q3		How much time does the tool need for its functions?
AU	M3.1	Time to create code dependencies
AU	M3.2	Number of dependencies found
AU	M3.3	False dependencies found
AU	M3.4	Time to create HTML data from code
AU	M3.5	Number of HTML code files
AU	M3.6	Number of HTML code lines
RN	M3.1	Number of files
RN	M3.2	Files not included
RN	M3.3	Number of entities found
RN	M3.4	Number of relationships
RN	M3.5	CatNice parsing time
RN	M3.6	Time to generate design
RN	M3.7	Time to clean database
RN	M3.8	Time to clean design
IAS	M3.1	RN TDIF size
IAS	M3.2	IAS data creation time
IAS	M3.3	Data loading time
IAS	M3.4	Number of statements
Q4		How does the tool support ease of use?
	M4.1	Input models value
	M4.2	Navigation value
	M4.3	Selection value
	M4.4	Component activation value
	M4.5	Application design principles value
	M4.6	Controls, groups and models value
Q5		Does the tool offer support for the novice user?
	M5.1	Context sensitive help
Q6		Does the tool offer support for the expert user?
	M6.1	Number of short-cuts
Q7		How is data integration supported?
	M7.1	Number of import formats from TDIF
	M7.2	Number of export formats to TDIF
	M7.3	Number of source-code lines supported
	M7.4	Number of document formats supported
	M7.5	Number of output formats

Table 2. *Continued*

Question	Metric	Definition of question or metric
Q8		What is the reliability of the tool?
	M8.1	Number of program crashes
	M8.2	Percentage of recoveries from crashes
	M8.3	Number of failures after user misuse
	M8.4	Percentage of recoveries from failures
	M8.5	Autosave ability
Q9		Does the tool have input data checking tools?
	M9.1	Number of fields having syntactic/semantic checking
	M9.2	Percentage of these fields
Q10		How can the tool behaviour be modified?
	M10.1	Number of options
Q11		What kind of information of its work does the tool offer?
	M11.1	Number of reports
Q12		What kind of tutorials is provided?
	M12.1	Number of on-line tutorial (training) documents
Q13		What is the documentation level of the tools?
	M13.1	Number of documents
	M13.2	Number of on-line documents
	M13.3	On-line index/search system

RN and IAS tools for Question 3. The other questions are common. In scoring the application-specific elements, we had to relate the metric to the application on which we did the tool assessment. The common elements are general mainly independent on the application software.

3.4. Metrics collecting

For the metrics collecting we used:

- standard UNIX tools,
- some specific tools, and
- guides for the definitions of metrics.

We obtained basic information about the tools and their performance during evaluation from UNIX utilities like `wc` and from the ReverseNICE statistics reports. The timing measurements were performed using the `clock` utility since it contains a timing clock. Even though it was not a particularly exact way of measuring time, it was adequate for our purposes. In most cases the measured times were long enough for manual timing. Resource/memory usage was measured using the `top` utility. It measures processes and their memory/system usage during given intervals. The user interface was evaluated using different kinds of calculations related to the user interface.

Metrics for 'ease of use' (Table 2, Question 4) were gathered from the style guide designed for the graphical user interface. The style guide for Motif (OSF, 1993, Appendix

B) was used for this purpose. That style guide contains a checklist with more than 230 checkpoint items to ensure style compatibility. Some 80 of the checkpoint items were not appropriate for our tool evaluation. Each used checkpoint item had three options: yes, no or not applicable. These checkpoint items were organized into six groups:

- input models,
- navigation,
- selection,
- component activation,
- application design principles and
- controls, groups and models.

Because there was a large number of checkpoint items, we decided to condense the list into a more practical form. To that end, we devised a procedure that gave from one to five points for each tool in each category. One point was given for each 20 per cent of checkpoint items containing the answer 'yes'. If, for example, 41 per cent of checkpoints in a specific group, say 'selection', contain the answer 'yes', then the metric M4.3 for selection value in 'ease of use' was recorded as two, and so on.

4. EVALUATION

4.1. Software used for tool evaluation

4.1.1. Applications profile

In the tool evaluation we used three different applications. The on-board space software METEGG developed by SSF was used, especially at the beginning of the tools development, to provide the first comments. Later, the slightly bigger on-board space software SIXA developed by VTT Electronics served as a test platform. The launching of the instruments including this software was at the end of 1996. The most active maintenance phase of this on-board software was not so heavy. Therefore, a third application, the software testing-tool MOSIM, was also used in our tool evaluation work.

These three applications provided the basis for our tools evaluation. Table 3 profiles them by giving statistical data about each.

The average number of source code lines per code file indicates that SIXA can be categorized as average size, METEGG as small size and MOSIM as large size software. Although the number of files is small in MOSIM, each file contains many lines of code. It is noteworthy that both SIXA and MOSIM contain more comments in the code than does METEGG. This reduces the differences in the lines of compileable code per file. In addition, there are some specific features in the code that distinguish the applications—e.g., the external header files of MOSIM added a lot of data to the ReverseNICE and IAS databases.

Table 3. Comparisons of application software used

Criterion	METEGG	SIXA	MOSIM	Notes
Code files	25	35	14	
Lines of code	8 023	38 421	55 364	
Average lines/file	320	1 098	3 954	
Comment lines	2 290	12 124	13 875	
Comment (%)	28.67	37.34	32.36	
Header files	38	50	11	MOSIM has 34 external header files
Assembler files	12	14 + 6 assembler headers	0	Not included in evaluation

4.1.2. Space application METEGG

The METEGG software was programmed at a very low level in a PC environment using the C and Assembler languages (Korpela, Kumpulainen and Harri, 1997). The software was designed using an object-orientated design method. A simple multitasking operating system for METEGG software was developed for the project and included, since that operating system probably cannot be used anywhere else. It has been customized to the software and hardware used.

The METEGG software runs separate processes for each instrument in the station. Every hardware component has a driver. Some of the instruments are connected directly to the processor while most of them are connected through a bus.

4.1.3. Space application SIXA

The SIXA on-board software is an embedded, concurrent real-time software system with strict timing requirements and tight coupling with the hardware of the instrument (Leppala *et al.*, 1996). The software is in the programming language ANSI C except for about 10 per cent of it which is coded in M68000 assembly language.

This on-board software handles X-ray photon data in real time, and stores the data to the RAM and hard disk of the instrument. The software also monitors and reports the status of the instruments. In-flight maintenance and recovery from some hardware faults have been built into the software.

4.1.4. Support software MOSIM

MOSIM is a simulation-based testing tool, which is used in a workstation environment during the integration and system testing of embedded software. MOSIM performs dynamic testing, which means that the real application to be tested is executed on a host. MOSIM provides utilities for simulating the operational environment of the application software, and possesses capture/playback and advanced test controlling features, such as the global time concept (Honka, 1992, pp. 70–98).

Since release 7 in 1993, the MOSIM software has been in the maintenance phase. Most

Table 4. Metrics from AU evaluation

Number	Metric	Value	Notes
M1.1	Disk space required for the tool	13.9 MB	
M1.2	Disk space required for additional tools	25 MB 1 MB	Webmaker Robochart
M1.3	Resource usage	1–2 MB during scripts; 1% CPU	Webmaker: 18–25 MB and 35–82% CPU Robochart: 2 MB
M1.4	Memory usage	1.3 MB	Webmaker: 240 KB Robochart: 684 KB
M1.5	Number of available platforms	1	SunOS, domain analysis part has Windows version
M2.1	Number of parameters needed to be configured	1	Assuming external tools are in path
M4.1	Input models value	3	Input is well supported for mouse. Focus is sometimes confusing
M4.2	Navigation value	4	Navigation is well supported, some problems with menu functionalities
M4.3	Selection value	2	There are only a few advanced features
M4.4	Component activation value	1	No mnemonics of short-cuts
M4.5	Application design principles value	1	Menus are not according to standards. The options are not shown via dialogue boxes
M4.6	Controls, groups and models value	3	Generally according to standards
M5.1	Context sensitive help	No	
M6.1	Number of short-cuts	0	
M7.1	Number of import formats from TDIF	0	
M7.2	Number of export formats to TDIF	0	
M7.3	Number of source codes supported	1	C source code
M7.4	Number of document formats supported	1	FrameMaker, through its import more than 10
M7.5	Number of output formats	2	HTML and graphs
M8.1	Number of program crashes	Some	Sometimes functions did not work properly; no actual crashes
M8.2	Percentage of recoveries from crashes	0%	No recovery mechanism
M8.3	Number of failures after user misuse	Several	
M8.4	Percentage of recoveries from failures	75%	Usually recovers, but may cause fatal problems
M8.5	Autosave ability	No	
M9.1	Number of fields having syntactic/semantic checking	10	
M9.2	Percentage of these fields	71%	
M10.1	Number of options	1	
M11.1	Number of reports	0	
M12.1	Number of on-line tutorial (training) documents	2	General and domain-analysis specific
M13.1	Number of documents	2	Specification and user guide
M13.2	Number of on-line documents	3	Training course, help
M13.3	On-line index/search system	0	

Table 5. Metrics for AU on the METEGG, SIXA and MOSIM applications

Number	Metric	METEGG	SIXA	MOSIM	Notes
M3.1	Time to create code dependencies	2 s	7 s	4 s	
M3.2	Number of dependencies found	208	784	412	
M3.3	False dependencies found	0(0%)	38(5%)	4(1%)	SIXA problems in IF statement and macro
M3.4	Time to create HTML data from code	18 s	59 s	76 s	
M3.5	Number of HTML code files	76	74	26	
M3.6	Number of HTML code lines	16 453	48 547	57 928	

of the maintenance work is involved in optimizing the performance of MOSIM software. Some work has been done to improve the test scripts and to create demos.

4.2. Results

4.2.1. Evaluation process

Our evaluation was based on the evaluation criteria described in Section 3.2. We used the metrics described in Section 3.3 to give guidance. The scoring (good, normal, low) was determined from our experiments on the three applications. This scoring was also influenced by related metrics and by feedback from the maintainers who used the tools during their maintenance work. As noted previously, our evaluation also included common as well as application-specific parts, which gave us additional insight.

4.2.2. Results from collected metrics

Table 4 shows a sample of the metrics gathered in the AU (application understanding) tool-set evaluation. Each metric was calculated, and the results were used to score the evaluation criteria. Table 5 shows metrics acquired from testing the performance of the AU tool-set with the METEGG, SIXA and MOSIM applications. Each table indicates metrics, results and some explanatory notes.

4.2.3. Criteria results

Each tool was rated on the seven criteria described in Section 3.2. The following results are a sample from the evaluation of the AU tool-set concerning the performance criterion.

- Response time: normal; reference metrics: M3.1 to M3.6.
  Response time of the AU tools was considered satisfactory. Webmaker was the only tool that required a slightly longer execution time. The graphical interface for the AU tools was created with Tcl/Tk. Because the Tcl/Tk is an interpretative language, response times were a bit longer than with interfaces created using translative languages.

- Installation: normal; reference metrics: M1.1, M1.5 and M2.1.
  Installation of the tools was straightforward because most of the tools were in the same directory. The AU tools did not check the operating system to locate third-party utilities. The system administrator or the user may install the separate tools.
- Start up: good; reference metric: M2.1.
  The AU tools required one directory definition before use. This definition was located in the script file that was run when the program was activated.
- Capacity: good; reference metrics: M1.1 to M1.4.
  There were no capacity restrictions, and the need for system resources was acceptable. The third-party utility, Webmaker, required somewhat more from the system.
- Efficiency: good; reference metrics: M1.3 to M.14.
  The AU tools used the system resources efficiently. See above comments about Webmaker.
- Tool portability: low; reference metric: M1.5.
  Tool portability was low since the AU tools only existed in the UNIX Sun operating system environment. Domain analysis tools can run in both Windows and UNIX environments. The common user interface for the AU tools only existed in the UNIX environment.

Appendix B contains the list of all the criteria. Each criterion was similarly scored and given textual comments and references to corresponding metrics.

5. CONCLUSION

Although this evaluation framework included a lot of uncertainty aspects, it provided a practical approach to obtain and document software tool evaluations. The definition of some metrics required assuming the presence of specific computer resources. However, this framework guaranteed a commensurable approach in spite of different tools and applications. In addition, there were several persons with different backgrounds doing the evaluation, and this framework provided a unifying method for gathering and evaluating results. From the point of view of finalizing project results, the tools evaluation done with this framework provided a workable basis for comparison of project results, and for relating them to each other.

APPENDIX A. GLOSSARY OF ACRONYMS AND NAMES

AMES	Application management environments and support, EU ESPRIT programme project 8156
AU:	Application understanding tool-set, part of the AMES project tools, developed by the University of Durham and VTT Electronics
CASE:	Computer-aided software engineering
CatNICE:	Code analysis tool, part of ReverseNICE tool, developed by Intecs Sistemi Spa
CPU:	Central processor unit

ESF/EPSOM:	European platform for software maintenance, 1991–1994; was a subproject of the Eureka Software Factory project (ESF) of the Eureka programme
ESPRIT:	European strategic programme for research and development in information technologies
EU:	European Comission
Eureka:	Pan-European, market-orientated research and development programme
Framemaker:	Text processing tool, developed by Frame Technology Inc.
GQM:	Goal, question and metrics
HOOD:	Hierarchical object-orientated design
HOOD/C:	Hierarchical object-orientated design and C source code generator
HTML:	HyperText mark-up language
IAS:	Impact analysis tool-set, part of the AMES project tools, developed by Matra Marconi Space France
MB:	Mega byte
METEGG:	On-board space application developed by SSF
MOSIM:	Real-time software testing tool developed by VTT Electronics
OS:	Operating system
ProcessWeaver:	Process enactment tool, developed by Cap Gemini Innovation
ReverseNICE:	Reverse-engineering tool, part of the AMES project tools, developed by Intecs Sistemi Spa
RN:	ReverseNICE
Robochart:	Flow-diagram editor, developed by Digital Insight Inc.
S:	Seconds
SIXA:	On-board space application developed by VTT
SunOS:	Operating system for Sun workstations, developed by Sun Microsystems Inc.
SSF:	Space Systems Finland
Tcl/Tk:	Scripting language for applications developed to run on Sun Microsystems environments
TC:	Tools criterion, used in tool evaluation
TDIF:	Traceability data input format
TEKES:	Technology Development Centre of Finland
UNIX:	Operating system, trademark of AT&T
VTT:	Technical Research Centre of Finland
Webmaker:	Framemaker cross-reference converter, developed by Harlequin Group Limited

APPENDIX B. TOOLS EVALUATION CRITERIA

The criteria are evaluated using three levels describing how the criteria are satisfied. These possible levels are low, normal and good. Some help for the evaluation is provided by defining satisfaction levels for each criterion as shown below for criteria performance (TC1), ease of use (TC2), tool integration/data exchange (TC3), tool robustness (TC4), functionality (TC5), training courses (TC6) and documentation (TC7).

TC1: Performance
- TC1.1: Response time
  - Low: user has to wait even in simple tasks
  - Normal: user has to wait in complex tasks
  - Good: tool is always waiting for the user
- TC1.2: Installation
  - Low: requires adaptation of the host system
  - Normal: requires some host system configuration and installation by a system administrator
  - Good: easy installation, can be performed by the user
- TC1.3: Start-up
  - Low: tool cannot be used without expert parametrization
  - Normal: some effort must be used to instantiate the tool into the user environment and use context
  - Good: simple start-up procedures
- TC1.4: Capacity
  - Low: capacity restrictions limit tool's normal use
  - Normal: capacity limits are reached only in exceptional cases
  - Good: no capacity restrictions
- TC1.5: Efficiency: the tool's ability to take advantage of system resources—i.e., memory, CPU, etc.—with respect to its functionality
  - Low: the tool uses a greater part of the system resources
  - Normal: the tool's use of system resources is acceptable with respect to its functionality
  - Good: the tool uses the system resources efficiently—e.g., it does not slow down other tools when in use
- TC1.6: Tool portability
  - Low: only available on one platform
  - Normal: available on several platforms among the most commonly used
  - Good: available on several platforms, minor adaptations needed to run in any platform

TC2: Ease of use
- TC2.1: Friendliness of the user interface
  - Low: the tool does not conform well to GUI standards (e.g., it is difficult to find/locate menu functions). Basic use of the tool requires training
  - Normal: the tool conforms to GUI standards, so that users easily recognize menu functions, etc.
  - Good: the tool combines GUI standards with advanced usage
- TC2.2: Ability to propose default values
  - Low: defaults are never proposed
  - Normal: defaults are occasionally proposed
  - Good: in every possible situation, pertinent default values are proposed
- TC2.3: Ability to use without comprehensive training
  - Low: tool can be used after comprehensive training
  - Normal: tool can be used after short training
  - Good: tool can be used without training

TC2.4: Ability to help the user on-line
- Low: the tool gives no on-line help
- Normal: the tool gives on-line help
- Good: the tool gives context sensitive on-line help

TC2.5: Ability to support novice
- Low: novice user cannot use the tool without help of expert user
- Normal: novice user can use tool by reading manual and/or using on-line help
- Good: novice user can instantly start using the tool

TC2.6: Ability to support expert user
- Low: no short-cuts available
- Normal: expert user can use keyboard short-cuts in most common tasks
- Good: expert user can do special routines, which runs tasks efficiently

TC2.7: Learning curve: ease of use and sophistication of tool with respect to learning to use the tool (not training)—i.e., facilitates becoming a skilled user
- Low: although the tool provides restricted functionality, it is difficult to become familiar with it
- Normal: the tool is sophisticated and easy to use, and requires some working hours to become familiar with it
- Good: the tool is sophisticated and requires few working hours to become an advanced user

TC8: Tool integration/data exchange

TC3.1: Level of integration
- Low: tool works as stand-alone
- Normal: tool has simple integration mechanisms, like invoking other tools or cut-and-paste possibility with other tools
- Good: tool is integrated (e.g., in the level of data, control and process) with other tools

TC3.2: Support for import
- Low: no data can be imported
- Normal: data in one defined format can be imported
- Good: data in several different formats can be imported

TC3.3: Support for export
- Low: no data can be exported
- Normal: data in one defined format can be exported
- Good: data in several different formats can be exported

TC4: Tool robustness

TC4.1: Reliability
- Low: the tool crashes frequently and/or exhibits unpredictable behaviour even under light usage and without user misuse
- Normal: the tool seldom crashes and/or exhibits unpredictable behaviour under heavy usage or user misuse
- Good: the tool never crashes and always exhibits predictable behaviour under heavy usage or user misuse

TC4.2: Data/error recovery
- Low: no mechanisms for error and data recovery

- Normal: automatic saving option. Files saved automatically must be detected by the user
- Good: tool saves the information automatically before system error and suggests the most recent non-defect-affected version to be used after recovery

TC4.3: User misuse/invalid data

- Low: tool crashes in the case of user misuse or invalid data
- Normal: tool gives error messages in the case of user misuse or invalid data
- Good: tool gives error messages and context sensitive help in the case of user misuse or invalid data

TC5: Functionality

TC5.1: Coverage

- Low: the tool provides one function
- Normal: the tool provides a set of related and integrated functions
- Good: the tool provides a complete family of facilities

TC5.2: Life-cycle/method coverage

- Low: the tool does not cover any defined life cycle/method properly—i.e., it only supports (part of) one activity
- Normal: the tool supports large parts of a defined life cycle/method, only some advanced concepts may be uncovered
- Good: the life cycle/method defined is covered comprehensively

TC5.3: Ability to support syntactic and semantic checking

- Low: no checking done by the tool
- Normal: only syntactic checking done
- Good: syntactic and semantic checking

TC5.4: Report facilities:

- Low: the tool provides no reports on the work which it performs
- Normal: the tool provides reporting facilities (warnings, diagnosis,…)
- Good: interactive reporting, the level of report can be adapted on demand

TC5.5: Tool flexibility

- Low: tool behaviour cannot be modified by the user
- Normal: tool behaviour can be modified to better meet user's need
- Good: tool behaviour can be comprehensively altered to achieve different goals

TC5.6: Ability to support team work

- Low: single user
- Normal: multi-user, but the users have to organize the work themselves
- Good: multi-user, the tool takes care of organization and protection

TC6: Training courses

- Low: no or low-level courses available
- Normal: normal level courses available
- Good: good courses for different kinds of user groups, skill levels, etc.

TC7: Documentation

TC7.1: Completeness

- Low: no or low-level user manual available
- Normal: useful user manual plus reference guide

- Good: complete and extensive well organized documentation

TC7.2: Ease of reference
- Low: no alphabetical index available
- Normal: table of contents and alphabetical index available
- Good: reference manual available

TC7.3: Accuracy
- Low: there are several mistakes or parts not covered in the documentation
- Normal: there are few mistakes/discrepancies towards the actual tool functions
- Good: there are no mistakes/discrepancies

TC7.4: Readability (language)
- Low: the text is poorly structured, many spelling errors, bad English, compact text
- Normal: the text is structured, contains few spelling errors, is well written (i.e., restricted use of acronyms, complicated/ambiguous words, etc.)
- Good: the text is well structured and presented in an 'easy-to-read' way

TC7.5: Presentation (layout and general 'feel')
- Low: no illustrations, different fonts, compact text, poor disposition
- Normal: illustrations in black and white, good disposition
- Good: colour illustrations, good disposition, 'professional feeling'

Acknowledgements

We thank the people who helped us to make the case studies. Mr. Kari Kumpulainen of SSF provided comments as a METEGG software developer and maintainer, Mr. Jukka Toivonen of VTT Electronics as a SIXA representative as well as Mr. Hannu Honka, Dr. Kari Laitinen, Mr. Juhani Latvakoski, Mr. Juha Lehtikangas and Mr. Mika Salmela of VTT Electronics as MOSIM developers and maintainers. In addition, Dr. Markku Oivo and Mr. Toni Sandelin provided comments on applying the GQM paradigm.

References

AMES (1994) *AMES Methodology and Process Model*, AMES Deliverable D111, Cap Gemini Innovation, Grenoble, France, 94 p.

Barros, S., Bodhuin, T., Escudie, A., Queille, J. P., and Voidrot, J. F. (1995) 'Supporting impact analysis: a semi-automated technique and associated tool', in *Proceedings International Conference on Software Maintenance*, IEEE Computer Society Press, Los Alamitos CA, pp. 42–51.

Boloix, G. and Robillard P. (1995) 'A software system evaluation framework', *IEEE Computer*, **28**(12), 17–26.

Basili, V. R. and Weiss, D. M. (1984) 'A methodology for collecting valid software engineering data', *IEEE Transactions on Software Engineering*, **SE–10**(6), 728–738.

Basili, V. R., Caldiera, G. and Rombach, H. D. (1994) 'Goal question metric paradigm', in Marciniak, J. J. (Ed), *Encyclopedia of Software Engineering*, Volume 1, John Wiley & Sons, Inc., New York NY, pp. 528–532.

Battaglia, M. and Savoia, G. (1997) 'ReverseNICE: a nice way to re-engineering legacy systems', in *Data Systems in Aerospace*, SP–409, ESA Publications Division, ESA/ESTEC, Noordwijk, The Netherlands, pp. 169–173.

Boldyreff, C., Burd, E. L., and Hather, R. M. (1994) 'An evaluation of the state of the art for application management', in *Proceedings International Conference on Software Maintenance*, IEEE Computer Society Press, Los Alamitos CA, pp. 161–169.

Boldyreff, C., Burd, E. L., Hather, R. M., Mortimer, R. E., Munro, M. and Younger, E. J. (1995) 'The AMES approach to application understanding: a case study', in *Proceedings International Conference on Software Maintenance*, IEEE Computer Society Press, Los Alamitos CA, pp. 182–191.

Harjani, D.-R. and Queille, J.-P. (1992) 'A process model for the maintenance of large space systems software', in *Proceedings Conference on Software Maintenance*, IEEE Computer Society Press, Los Alamitos CA, pp. 127–136.

Haziza, M., Voidrot, J. F., Minor, E., Pofelski, L., and Blazy, S. (1992) 'Software maintenance: an analysis of industrial needs and constraints', in *Proceedings Conference on Software Maintenance*, IEEE Computer Society Press, Los Alamitos pp. 18–26.

Honka, H. (1992) *A Simulation-based Approach to Testing Embedded Software*, available as VTT Publications 124, Technical Research Centre of Finland, VTT Electronics, Espoo, Finland, 118 pp.

Kitchenham, B., Pickard, L. and Pfleeger, S. L. (1995) 'Case studies for method and tool evaluation', *IEEE Software*, **12**(4), 52–62.

Kitchenham, B. A. (1996) 'Evaluating software engineering methods and tools, part 3: selecting an appropriate evaluation method—practical issue', *Software Engineering Notes*, **21**(4), 9–12.

Korpela, S., Kumpulainen, K. and Harri, A.-M. (1997) 'Autonomous software of the MARS'96 small stations', in *Data Systems in Aerospace*, SP-409, ESA Publications Division, ESA/ESTEC, Noordwijk, The Netherlands, pp. 157–162.

Laitinen, K. (1995) 'Natural naming in software development and maintenance', Doctoral Dissertation, available as *VTT Publications 243*, Technical Research Centre of Finland, VTT Electronics, Espoo, Finland, 99 pp. + 70 pp. appendix.

Leppala, K., Korhonen, J., Ruuska, P., Toivanen, J. and Paivike, H. (1996) 'Real-time approach for development of scientific space instrument software', in *Proceedings Eighth Euromicro Workshop on Real-time Systems*, IEEE Computer Society Press, Los Alamitos CA, pp. 139–144.

Makarainen, M. (1996) '*Application management requirements of embedded software*', Licentiate Thesis, available as *VTT Publications 286*, Technical Research Centre of Finland, VTT Electronics, Espoo, Finland, 99 pp.

NASA (1995) '*Software measurement guidebook, revision 1*,' Technical Report SEL–94–102, NASA Goddard Space Flight Center, Greenbelt MD, 134 pp.

Oivo, M. (1994) '*Quantitative management of software production using object-oriented models*', Doctoral Dissertation, available as *VTT Publications 169*, Technical Research Centre of Finland, VTT Electronics, Espoo, Finland, 72 pp. + 70 pp. appendix.

OSF (1993) *OSF/Motif Style Guide Release 1.2, Open Software Foundation*, Prentice-Hall International, Englewood Cliffs NJ, 400 pp.

Robinson, P. (1992) *HOOD Hierarchical Object-oriented Design*, Prentice-Hall International, Hertfordshire, UK, 238 pp.

Taramaa, J., Makarainen, M. and Ketola, T. (1995) 'Improving application management process through qualitative framework', in *Proceedings International Conference on Software Maintenance*, IEEE Computer Society Press, Los Alamitos CA, pp. 327–336.

Taramaa, J. and Ketola, T. (1995) 'Process modelling and software maintenance aspects from the point of view of subcontractor in a space application', in *Proceedings of an International Symposium on On-board Real-time Software*, SP-375, ESA Publications Division, ESA/ESTEC, Noordwijk, The Netherlands, pp. 145–152.

Taramaa, J., Seppanen, V. and Makarainen, M. (1996) 'From software configuration to application management—improving the maturity of the maintenance of embedded software', *Journal of Software Maintenance*, **8**(1), 49–75.

van Latum, F., van Solingen, R., Oivo, M., Hoisl, B., Rombach, H. D. and Ruhe, G. (1998) 'Adopting GQM-based measurement in an industrial environment', *IEEE Software*, **15**(1), 78–86.

Vierimaa, M. (1996) '*A hypertext approach to application understanding: a maintenance perspective*', Master's Thesis, available as *VTT Publications 274*, Technical Research Centre of Finland. VTT Electronics, Espoo, Finland, 76 pp.

Authors' biographies:

Matias Vierimaa is a research scientist at VTT Electronics, where he has been in charge of several process improvement projects for software engineering environments for embedded software. His main areas of research interest include software process improvement, software measurements, and software maintenance environments for embedded real-time systems. He has an M.Sc. in Information Processing Science from the University of Oulu. His e-mail address is: Matias.Vierimaa@vtt.fi

Jorma Taramaa is a senior research scientist at VTT Electronics where he has been in charge of several projects developing methods and tools for software engineering environments for embedded software. His main areas of research interest include software process improvement, especially software maintenance, and software configuration management processes, as well as software engineering environments for embedded real-time systems. He has a Ph.Lic. and M.Sc. in Information Processing Science from the University of Oulu. His e-mail address is: Jorma.Taramaa@vtt.fi

Heli Puustinen is a programmer at VTT Electronics, where she has worked on projects developing tools for software engineering environments for embedded software. The main subject of the tools has been software configuration management. She graduated as a programmer at The Institute of Information Technology (IIT) in Espoo and is currently a student at the Department of Information Processing Science at the University of Oulu. Her e-mail address is: Heli.Puustinen@vtt.fi

Kati Suominen is currently working as a product manager for electronic commerce at EUnet International, a pan-European backbone Internet service provider. Before joining EUnet she worked at Space Systems Finland Ltd., a Finnish space-domain software company, involved in projects concerning ground segment software and application management. She is finalizing her Master's thesis at the Technological University of Helsinki. Her e-mail address is: kss@eunet.fi

Tommi Ketola is currently working as a senior software engineer at Teleste Ltd., a European professional broadband communication systems company. Before joining Teleste Ltd., he worked at Space Systems Finland Ltd., being in charge of several on-board software, ground segment software and application management projects. He has an M.Sc. in Electrical Engineering from the Technical University of Helsinki. His e-mail address is: Tommi.Ketola@teleste.fi

